# Automating GUI testing for SUMO Netedit

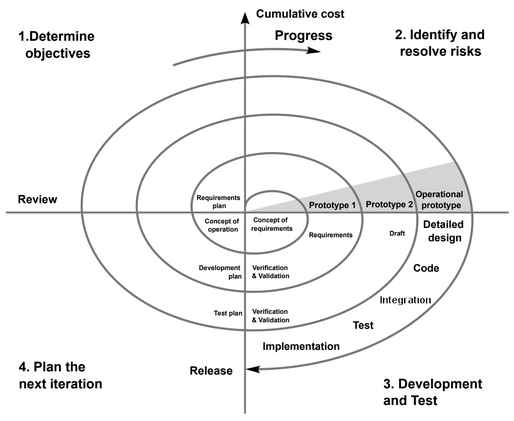
Pablo Álvarez López (pablo.alvarezlopez@dlr.de), Jakob Erdmann (jakob.erdmann@dlr.de)
Institute of Transportation Systems, German Aerospace Center (DLR), Berlin

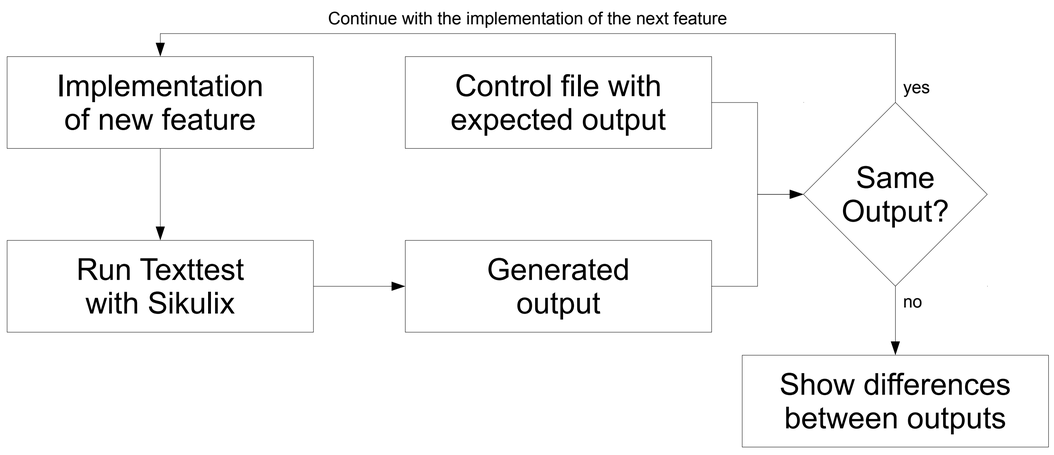
## Abstract

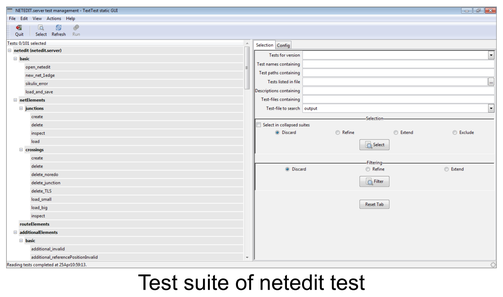
Verification is a critical step in the development of new Software[1], especially for Software that requires a Graphical User Interface (GUI) as well as a high grade of reliability. Currently, there exist several private and free tools for automating software testing, but only a small number of them support GUI Testing[4], and only for certain GUI-Frameworks. GUI testing should cover all functionality of the software (Domain) that usees a GUI as mechanism for data input and output using regression tests[2]. In most cases a tester or a group of testers is a basic requirement, and these tests are a series of simple steps (Mouse Clicks, Key presse, etc..) that generates a visual or written output. These steps can be achieve using Mouse position capture, Event capture or Screen capture. One of the more flexible tools for this GUI automatization is Sikulix[5]. This testing framework supports all major platforms and is independent from GUI Frameworks and accessibility APIs. Instead it uses image recognition powered by OpenCV to identify and control GUI components. In our use case this is a critical advantage because SUMO uses the FOX Toolkit as a GUI Framework which lacks build-in GUI Testing support.. SUMO already uses TextTest, a tool for automated testing of programs written in almost any programming language, and whose operation is based on comparing generated output files generated with previously defined control files.

## Sikuli Script and Texttest

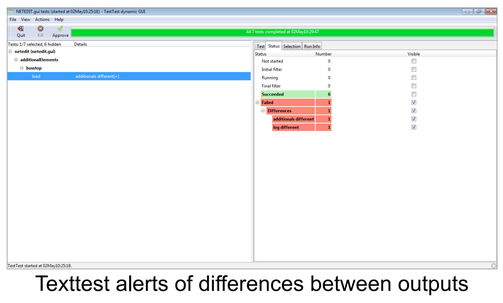
Regression testing verifies that previously developed and tested software performs correctly after a new change and ensure that changes during the different development phases don't introduce new faults[2]. Its very usefull for the process model based on a spiral[3]. Each cycle must be accompanied by a set of tests that will be run after the implementation of each new feature, to ensure that the new implementation doesn't cause a bug in other parts of the software.



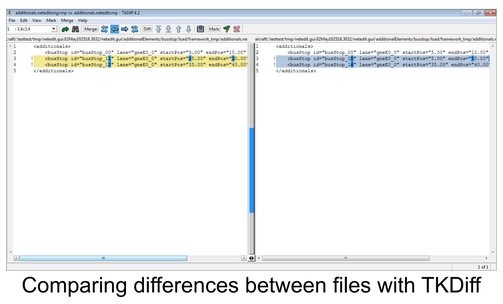



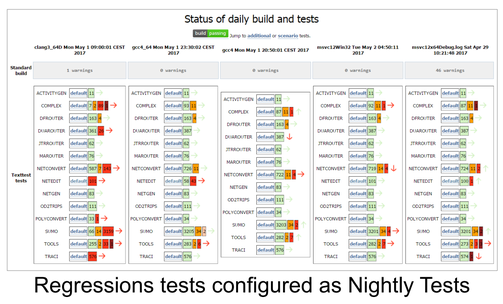
Test suite of netedit test



Texttest alerts of differences between outputs



Comparing differences between files with TKDiff



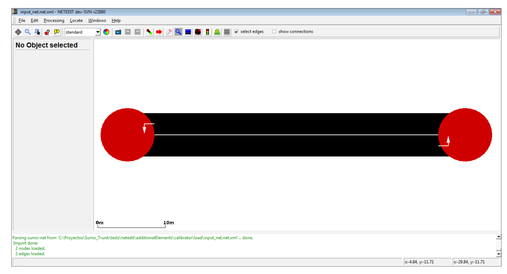

Regressions tests configured as Nightly Tests

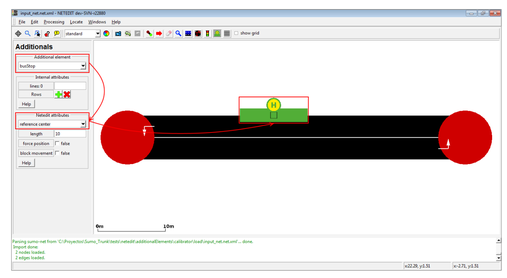
## Testing netedit using Sikuli

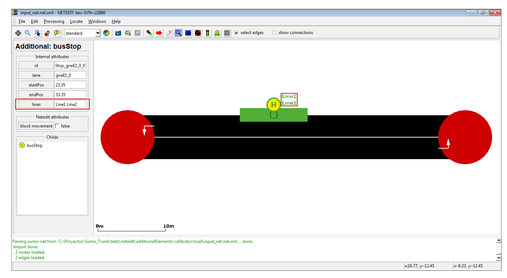
A regression test based on Sikuli consists of a python file that is executed through Texttest. Aditionally a previously defined input with nets, additionals, etc can be added. Testing actions can be carried out with calls to simple functions like t*ype*(string) for typing text or to press one or several keys at the same time, *click*(x,y) for clicking on specific map coordinates or *find*(image) for searching an image within the screen. The use of python to write test gives the possibility of using a large number of libraries to analyze the inputs and outputs obtained through the Sikuli functions, as well as being able to encapsulate a set of steps in a single function. After finishing the test, netedit saves the different output files (net, additionals, etc.), and they are automatically compared by Textest[5].

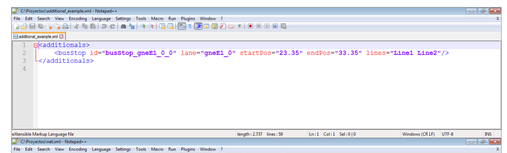
### Creation and inspection of a BusStop in Netedit using Sikuli Script

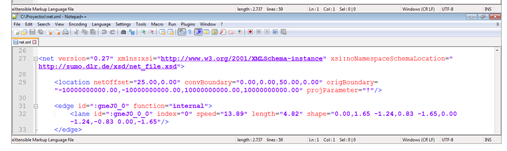
```
# import python libraries for netedit tests
import neteditTestFunctions as netedit

# Open netedit using a previously defined function
NeteditProcess = netedit.setupAndStart()

# go to additional mode pressing 'a' key
type("a")

# jump to comboBox with the list of additionals
type(Key.TAB)

# select current additional with Control + a
type("a", Key.CTRL)

# paste string "busStop" to select it
paste("busStop")

# type enter to save change
type(Key.ENTER)

# change reference to center jumping tusing tabs
for x in range(0, 9):
    type(Key.TAB)

# select current reference value
type("a", Key.CTRL)

# paste new reference
paste("reference center")

# create busStop clicking in position 425, 250
click(425, 250)

# go to additional mode pressing 'i' key
type("i")

# inspect first busStop
click(425, 255)

# Change lines jumping to their text field using tabs
for x in range(0, 5):
    type(Key.TAB)

# select current lines with Control + a
type("a", Key.CTRL)

# paste the new lines
paste("line1 line2")

# save additionals mit Control + Shift + d
type("d", Key.CTRL + Key.SHIFT)

# save network wtih
type("s", Key.CTRL)

# quit netedit using a previously defined function
netedit.quit(neteditProcess)
```



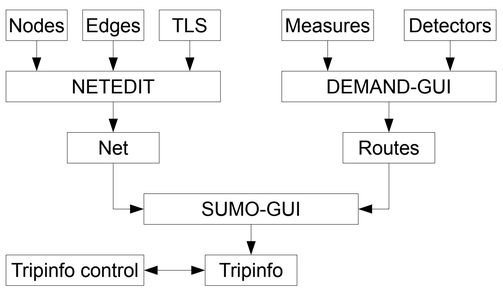





## Future

In successive revisiones, GUI tests will be extended to the rest of future SUMO-GUI components (DFROUTER-GUI, NETCONVERT-GUI, etc ...). The different componentswill be concatenated to test a complete simulations, starting with the basic components of a network and a traffic information until the final tripinfo output file. The output will be compared with the control output files using Texttest.




**References:**

[1] Sommerville, I. (2016). Software engineering. Pearson, 1, 8.

[2] Myers, G. J., Sandler, C., & Badgett, T. (2011). The art of software testing. John Wiley & Sons.

[3] Boehm, B. W. (1988). A spiral model of software development and enhancement. Computer, 21(5), 61-72.

[4] Singh, I., & Tarika, B. (2014). Comparative analysis of open source automated software testing tools: Selenium, sikuli and watir. International Journal of Information and Computation Technology, 4, 15.

[5] Yeh, T., Chang, T. H., & Miller, R. C. (2009, October). Sikuli: using GUI screenshots for search and automation. In Proceedings of the 22nd annual ACM symposium on User interface software and technology (pp. 183-192).

DLR Deutsches Zentrum für Luft- und Raumfahrt